به نام خدا

تمرین عملی شماره یک

مبانی بازیابی اطلاعات و جستجو وب

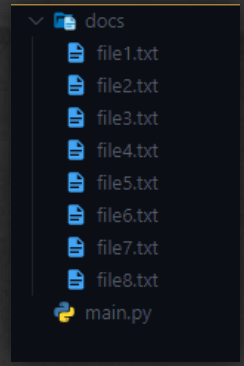
دانیال بیاتی ۹۸۱۶۰۷۳

محمد جواد کفایتی ۹۸۲۳۷۹۳

استاد : دکتر هدی مشایخی

برنامه ای بنویسید که تعدادی از اسناد به دلخواه دریافت کند و پرس و جو هایی را از کاربر دریافت کرده شامل **AND** و **OR** و **NOT** و به روش ایندکس معکوس اسناد مرتبط با آن را بازگرداند . اسناد را میتوانید از مجموعه داده های متنی از مخزن های متداول وب مثل **Kaggle** و **UCI** برداشت کنید. تعداد سندها نیازی نیست که زیاد باشد، و انجام کارهای پیش پردازش در این بخش نیازی نیست .نتایج پرس و جو را تحلیل کرده و درستی آنها را بررسی کنید.

داخل پوشه **docs** حاوی ۸ داکیومنت است که به صورت **file1 ... file8** نام گذاری شده است که شامل متن هایی به صورت زیر است :



docs > file1.txt

```
1  I'm looking for information on the passing of author and screenwriter Nora Ephron.
```

docs > file5.txt

```
1  A friend is an identical twin. Her sister is already showing signs of dementia at 65.
2  Now her twin is concerned she too will experience early onslaught of the disease.
3  I would like to know more about the disease focused on females.
4  Are there things we can do to lessen our chances of getting the disease?
5  Is the disease always identified by genetic testing?
```


01

فایل **main.py** برنامه اصلی می باشد که از دو تابع ساخت ایندکس و سرچ کوئری قرار دارد.

```python
def create_inverted_index(files):
    inverted_index = {}
    for i, file in enumerate(files):
        with open(file, 'r') as f:
            contents = f.read()
            words = contents.split()
            for word in set(words):
                if word not in inverted_index:
                    inverted_index[word] = []
                inverted_index[word].append(i + 1)
    for word in inverted_index:
        inverted_index[word].sort()
    return inverted_index


files = glob.glob('./docs/*.txt')
inverted_index = create_inverted_index(files)
# print(inverted_index)
```

تابع **create_inverted_index** :

به ترتیب ابتدا محتوای فایل های ورودی را خوانده و به کلمات را به صورت یک آرایه به نام **words** در می آوریم.

سپس اگر کلمه وجود نداشت در دیکشنری آن کلمه را اضافه می کنیم در غیر این صورت ایندکس فایل را که آن کلمه در آن موجود است را به کلمه مورد نظر در دیکشنری اضافه می کنیم این کار را تاجایی ادامه می دهیم که دیگر فایلی برای خواندن نداشته باشیم.

سپس هر کلمه در دیکشنری را با توجه به شماره فایل مرتب می کنیم

```python
def search_query(inverted_index, query):
    query_tokens = query.split()
    result_docs = set(range(1, len(files) + 1))

    # Loop through each token in the query
    for i in range(len(query_tokens)):
        token = query_tokens[i]

        # If the token is "and"
        if token == "and":
            # Get the intersection of the current result document set and the next token's document set
            next_token = query_tokens[i+1]
            if next_token not in inverted_index:
                return []
            next_docs = set(inverted_index[next_token])
            result_docs = result_docs.intersection(next_docs)

        # If the token is "or"
        elif token == "or":
            # Get the union of the current result document set and the next token's document set
            next_token = query_tokens[i+1]
            if next_token not in inverted_index:
                continue
            next_docs = set(inverted_index[next_token])
            print(result_docs)
            result_docs = result_docs.union(next_docs)
            print(result_docs)

        # If the token is "not"
        elif token == "not":
            # Remove the documents from the current result document set that are also in the next token's document set
            next_token = query_tokens[i+1]
            if next_token not in inverted_index:
                continue
            next_docs = set(inverted_index[next_token])
            result_docs = result_docs.difference(next_docs)

        # If the token is not an operator, it must be a search term
        else:
            # Get the document set for the current token and update the result document set accordingly
            if i - 1 >= 0 and (query_tokens[i - 1] == 'not' or query_tokens[i - 1] == 'or'):
                continue
            if token not in inverted_index:
                return []
            docs = set(inverted_index[token])
            result_docs = result_docs.intersection(docs)

    # Convert the final result document set to a sorted list and return it
    result_list = sorted(list(result_docs))
    return result_list
```

تابع **search_query** :

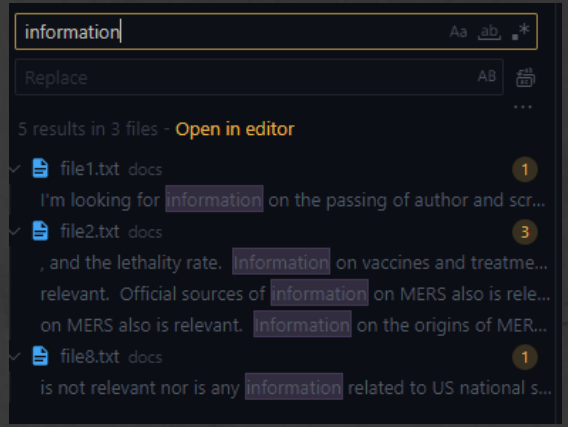
تابع **search_query** یک متن دریافت می‌کند و با استفاده از فهرست پوشا، اسنادی را که حاوی تمام کلمات موجود در متن هستند، پیدا می‌کند.

در ابتدا، متن ورودی به کلمات تقسیم شده و سپس برای هر کلمه، مجموعه اسنادی که آن کلمه در آن‌ها وجود دارد، با استفاده از فهرست پوشا به دست می‌آید. سپس با استفاده از عملگرهای منطقی **and** ، **or** و **not** ، مجموعه اسناد نهایی به دست می‌آید و به صورت لیست مرتب شده‌ای از اسناد بازگشت داده می‌شود.

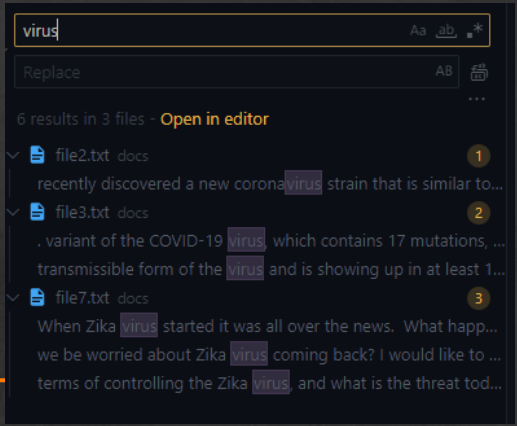
اگر کلمات جستجو شده در فهرست پوشا وجود نداشتند، خروجی خالی برگشت داده می‌شود.

```python
files = glob.glob('./docs/*.txt')
inverted_index = create_inverted_index(files)
# print(inverted_index)

query = input("Please enter query : ")
print(search_query(inverted_index, query))
```




```
Please enter query : not information and virus
[3, 7]
```



تست و بررسی داکیومنت ها :

برای مثال اگر کوئری زیر را داشته باشیم:

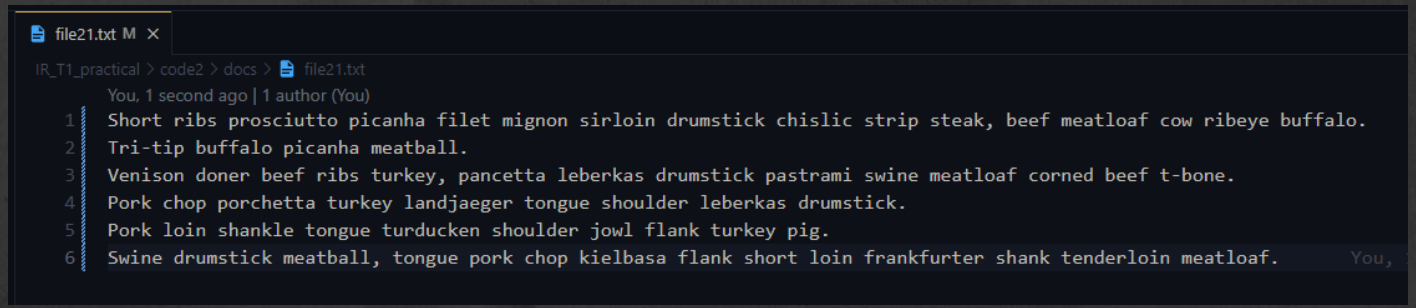
not information and virus

خروجی داکیومنت های ۳ و ۷ رانشان می دهد

همچنین با توجه به اینکه کلمه information در داکیومنت های ۱ و ۲ و ۸ وجود دارد لذا در داکیومنت ۳ و ۴ و ۵ و ۶ و ۷ وجود ندارد که اشتراکش با کلمه virus که در داکیومنت های ۳ و ۷ وجود دارد همان ۳ و ۷ می باشد.

فایل های متنی تمرین قبل را در تعداد بیشتر به وسیله نرم افزار **Lucene** ایندکس کنید. همه حروف را کوچک کنید و کلمات ایست را حذف کنید. ریشه یابی را به دلخواه خود می توانید انجام دهید. سپس حدقل ۵ پرس و جو مطرح کنید که عملگرهای **not, and, or** و جستجوی عبارت و پرس و جوی **wildcard** را شامل شوند و نتایج را بررسی کرده و گزارش کنید که آیا نتایج و رتبه بندی آنها منطقی هستند یا خیر.

داخل پوشه **docs** حاوی **30** داکیومنت است که به صورت **file1 ... file30** نام گذاری شده است که شامل متن هایی به صورت زیر است :



فایل **main.py** برنامه اصلی می باشد که از کتابخانه **whoosh** مثل **lucene** استفاده شده است که از توابع پیشفرض برای نرمالایز کردن داکیومنت ها استفاده شده است.

تابع **create_index_directory** : این تابع یک پوشه برای نگهداری ایندکس های ساخته شده می سازد.

تابع **create_schema** : این تابع از توابع پیشفرض در **whoosh** استفاده کرده و عملیاتی چون **stemming** و حذف **stopwords** ها را برعهده دارد که ایندکس های نهایی به صورت نرمالایز شده در پوشه ایندکس قرار می گیرد.

```
def create_index_directory(path="" , name="index"):
  if not os.path.exists(path+"index"):
      os.mkdir(path+"index")
  return  create_in("index", schema)

def create_schema():
  stopwords = frozenset(stoplists["en"])

  analyzer = StemmingAnalyzer(stemfn=stem, stoplist=stopwords) | StopFilter(stoplist=stopwords)

  return  Schema(title=TEXT(stored=True), path=ID(stored=True), content=TEXT(analyzer=analyzer))
```

تابع **indexing_doc** : این تابع فایل های موجود در پوشه **docs** را به ترتیب ایندکس می کند و به عنوان یک **writer** فایل های ایندکس شده را برای جستجو و تحلیل **commit** می‌کند.

```python
def indexing_docs(docs_addresses):
  writer = ix.writer()

  for i, file in enumerate(files):
      with open(file, 'r') as f:
        contents = f.read()

      writer.add_document(title=file, path= file , content=contents)

  writer.commit()


def searcher(boolean_query):
  #for create parser and return docs number
  with ix.searcher() as searcher:
      parser = QueryParser("content",  termclass=query.Variations ,schema=schema)
      q = parser.parse(boolean_query)
      results = searcher.search(q)
      for i , r in  enumerate(results):
        title = r["title"]
        print(f"score: {i+1}          -----       doc name: {title}")

```

تابع **searcher** : تابع **searcher** دارای پارامتر **boolean_query** می باشد که عبارت مورد نظر را دریافت می کند و به صورت پارس شده در جستجوگر قرار می دهد. این تابع با استفاده از **QueryParser** و **searcher** اطلاعات مورد نظر را به دست می آورد و شماره سند مورد نظر را بر می گرداند.

سپس با تست ۵ **query** صحت برنامه را امتحان می کنیم:

```python
global schema , ix
files = glob.glob('./docs/*.txt')
schema = create_schema()
ix = create_index_directory()
indexing_docs(files)

query_input = input("Enter query : ")
searcher(query_input)
```

```
23:07:51 ❯ py main.py
Enter query : starT OR HEllo
score: 1          -----       doc name: ./docs\file24.txt
score: 2          -----       doc name: ./docs\file7.txt
score: 3          -----       doc name: ./docs\file5.txt
score: 4          -----       doc name: ./docs\file25.txt
score: 5          -----       doc name: ./docs\file16.txt
score: 6          -----       doc name: ./docs\file29.txt
score: 7          -----       doc name: ./docs\file21.txt
```

```
23:12:40 ❯ py main.py
Enter query : HEllo AND reading
score: 1          -----       doc name: ./docs\file5.txt
```

```
23:13:30 ❯ py main.py
Enter query : relevant AND NOT virus
score: 1          -----       doc name: ./docs\file2.txt
score: 2          -----       doc name: ./docs\file19.txt
score: 3          -----       doc name: ./docs\file18.txt
score: 4          -----       doc name: ./docs\file14.txt
score: 5          -----       doc name: ./docs\file8.txt
score: 6          -----       doc name: ./docs\file17.txt
score: 7          -----       doc name: ./docs\file11.txt
```

```
23:11:39 ❯ py main.py
Enter query : HELLO AND NOT READING
score: 1          -----       doc name: ./docs\file25.txt
score: 2          -----       doc name: ./docs\file16.txt
score: 3          -----       doc name: ./docs\file29.txt
score: 4          -----       doc name: ./docs\file21.txt
```

```
23:10:27 ❯ py main.py
Enter query : Now AND ReaD
score: 1          -----       doc name: ./docs\file17.txt
score: 2          -----       doc name: ./docs\file5.txt
```